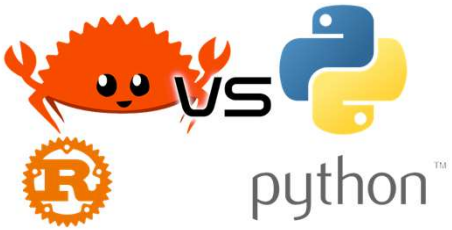
zelis

# Quantitative Comparison

Who wins in the race of time and space?




17th October 2025

## Contents

## Introduction

Quantitatively assess who does better in the race of time and space.

### Python

- Interpreted Language
- Ease of Use
- **Memory Management (**Uses automatic garbage collection)
- Exceptionally rich library support for **data science**
- Faster development and experimentation cycles
- Parallelism(Constrained by GIL)
- Error Handling

### Rust

- Compiler Language
- Suitable for performance-critical applications
- Uses an **ownership model** for safe and deterministic memory handling.
- Enforces strong compile-time checks
- Concurrency
- Growing ecosystem (notably **Polars**, **ndarray**, **Arrow** for data processing).
- Best suited for high-performance

# Experimental Setup

We are assessing performance and memory in two scenarios

### Preprocessing

- Detect Column types
- Missing value detection
- Imputation
- Normalization
- Add / Drop Columns
- Filter rows
- Sorting
- Sampling

### Searching Algorithms

- Linear Search
- Binary Search
- Interpolation Search
- Jump Search

## Preprocessing

Dataset size

- Healthcare claims dataset
- Size of Data : (116352, 32)
- ~14MB in size

## Searching Algorithms

Array of size 1 million within a range MIN and MAX

Statistics obtained on scenarios searching

- First
- Middle
- Last
- Element < MIN
- Element > MAX

## Time profiling

- Python - line profiler
- Rust – Measuring Elapsed time
- Hyperfine - Benchmarking for execution time

## Memory Profiling

- Python – memory_profiler
- Rust – using sysinfo

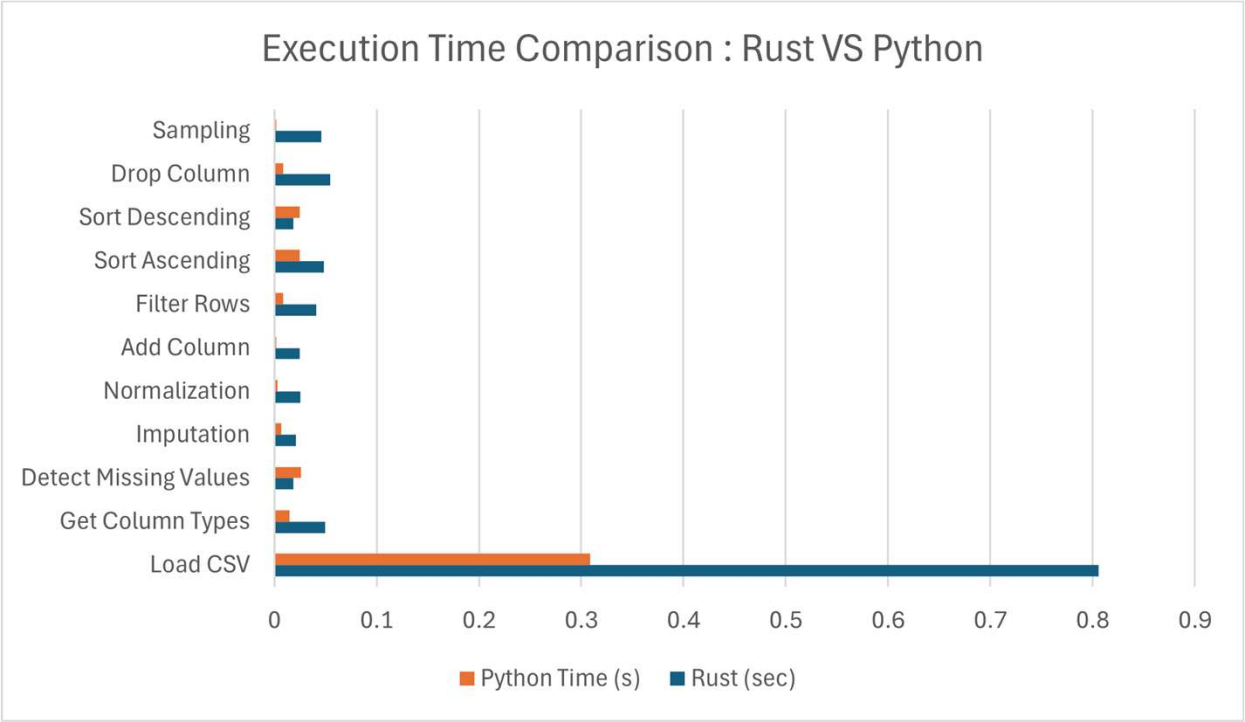
# Observations & Results

## Preprocessing

## Time Profiling Comparison

Python wins at each individual task execution time

- Loading dominated in both
- Python dominates in execution time per operation
- Visualizations shows Python's optimization and efficiency in data operations



## Benchmarking tells us a different story

Warmup Impact

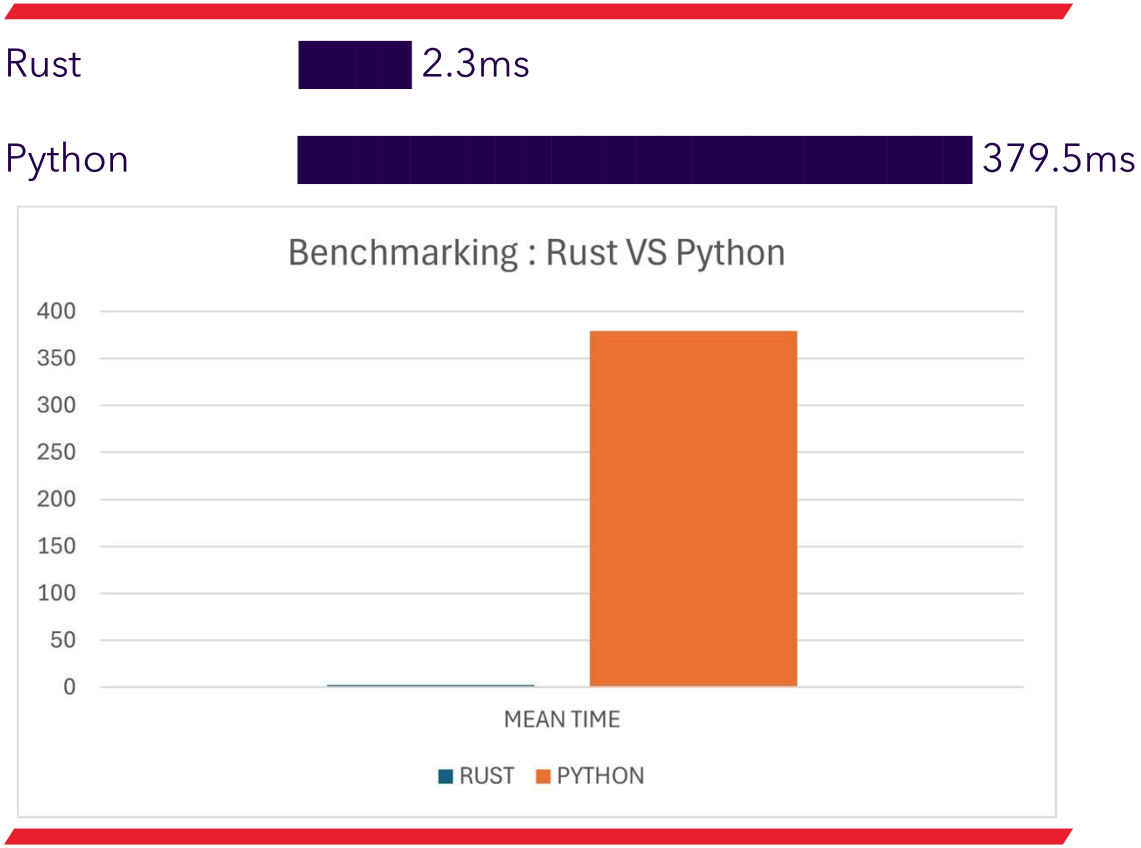
Stability

- Warmup (3 runs): Rust stabilized instantly, Python took ~0.39s.
- Warmup (10 runs): Rust stable at 0.00s, Python stable around 0.38s.

Improvement due to warmup

- Rust : 44%
- Python : 4%

This shows that the reason for slow per-operation execution is due to cache misses in course of process execution



# Memory Profiling

Rust consistently uses less memory than Python

Average memory footprint

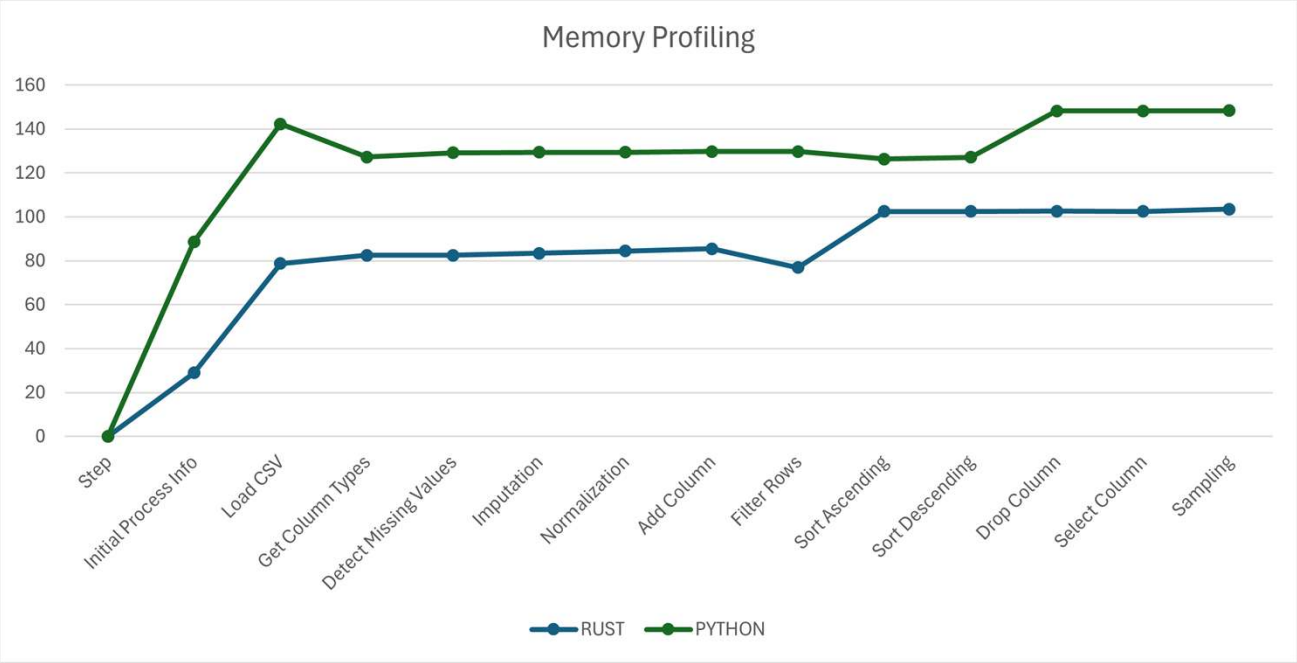
- Rust ~88 MB
- Python ~134 MB

Peak Memory

- Rust : 103MB
- Python : 148MB

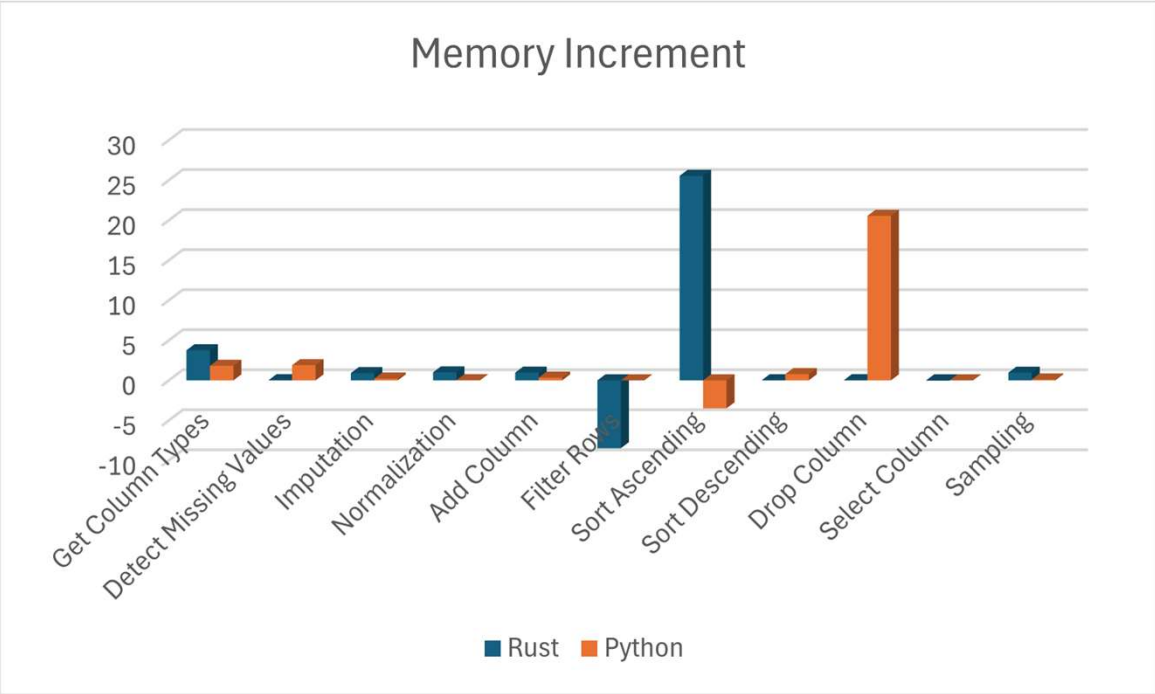
What is the use ?

More predictable **scaling for large datasets**.



Rust shows controlled memory reuse (drops after filtering), while Python retains buffers.

- Could be the result of selected rows getting dereferenced.(Ownership & Borrowing)



# Observations & Results

# Searching Algorithms

## Time Profiling

We observe that Python performs significantly better than Rust

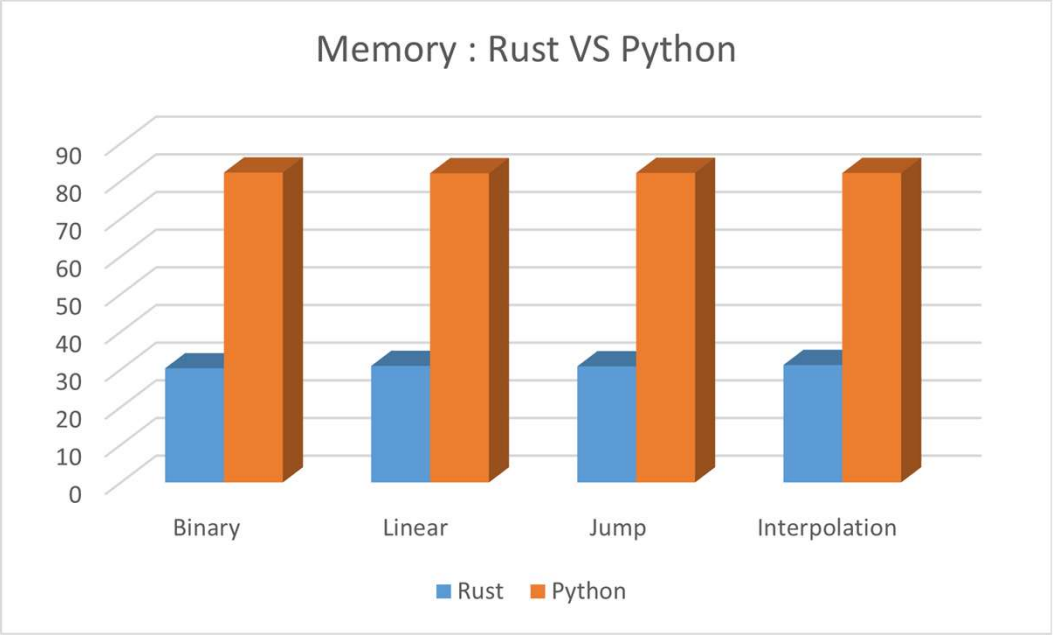
These are results of benchmarking on hyperfine

- There is no help even after warmup

| | Rust | Python |
|---|---|---|
| **Search Algorithm** | **Time sec** | **Time sec** |
| Binary | 1.124155 | 0.26 |
| Interpolation | 1.07282 | 0.32126 |
| Jump | 1.024465 | 0.321994 |
| Linear | 1.041261 | 1.04 |

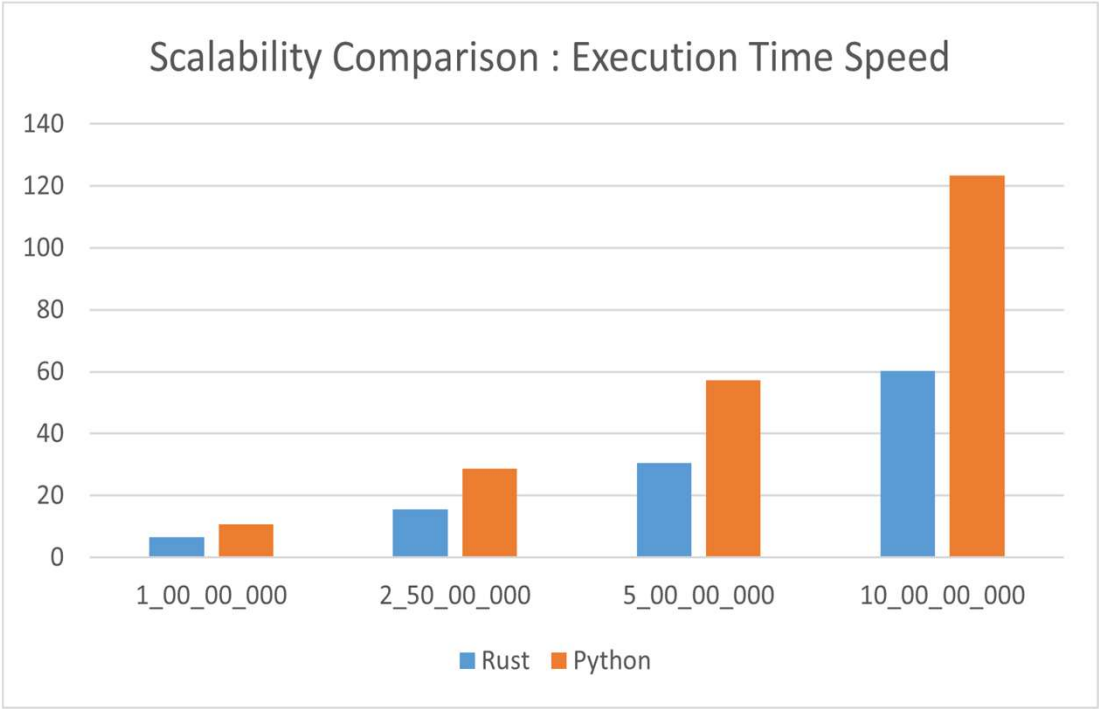
# Memory Profiling

- **Rust is much more memory-efficient** – roughly **2.7× less memory** than Python across all algorithms
- The **difference across algorithms** within the same language is minimal
- The **variation** (max–min) within:
  - Rust: ~0.9 MB (very stable)
  - Python: ~0.2 MB (also stable)



# Scalability

- What happens when we increase array size?
- Who scales better?
- Consider the search algorithm with max time complexity
- The comparative study shows that **Rust consistently outperforms Python** in both **execution time** and **memory efficiency**



# Conclusions

- **Time Performance**
  - **A. Rust outperforms Python** in both preprocessing and searching speed.
  - B. Shows **better scalability** and is ideal for **high-performance ETL** and **real-time data processing**.
  - **C. Python** remains stronger for **rapid prototyping** and **flexibility**.

- **Memory Performance**
  - **A. Rust uses less and more predictable memory** across all tasks.
  - B. Its **ownership model** ensures efficient reuse, unlike Python's **garbage collection**.
  - **C. Rust offers stable, scalable performance**, while **Python suits analytical experimentation**.

## Next Steps

- Integrate **multi-threading and parallelism** in Rust and **multiprocessing in Python** to evaluate performance gains and CPU utilization under concurrent workloads.

- Expand the scope to other algorithmic domains such as **sorting, regression, and clustering** to generalize the performance trends observed.

Pranav Phanindra Sai Manepalli

pranavphanindrasai.manepalli@zelis.com

7981145597